Chromium: A Stream Processing Framework for Interactive Rendering on Clusters

Greg Humphreys, Mike Houston, Ren Ng
Stanford University

Sean Ahern, Randall Frank
Lawrence Livermore National Laboratories

Peter Kirchner, James T. Klosowski
IBM T.J. Watson Research

The Problem

Scalable graphics solutions are rare and expensive

Commodity technology is getting faster

But it tends not to scale

Cluster graphics solutions have been inflexible

Why Clusters?

Commodity parts
- Complete graphics pipeline on a single chip
- Extremely fast product cycle
- More feature innovation

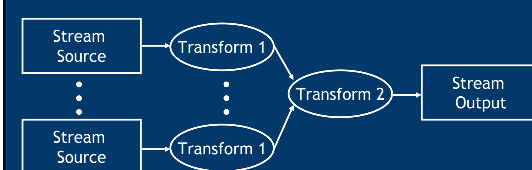
Flexibility
- Configurable building blocks

Cost
- Driven by consumer demand
- Economies of scale

Stream Processing

Streams:
- Ordered sequences of records
- Potentially infinite

Stream Transformations:
- Process only the head element
- Finite local storage



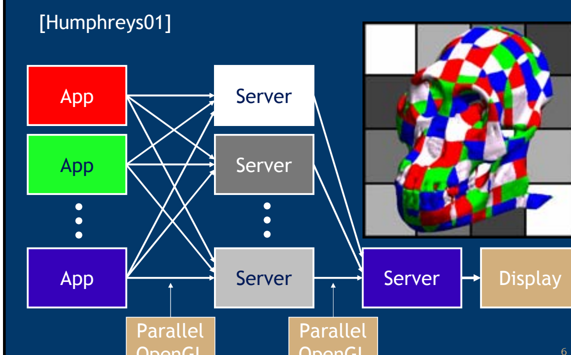
Why Stream Processing?

Elegant mechanism for dealing with huge data
- Explicitly expose and exploit parallelism
- Hide latency

State of the art in many fields:
- Databases [Terry92, Babu01]
- Telephony [Cortes00]
- Online Algorithms [Borodin98,O'Callaghan02]
- Sensor Fusion [Madden01]
- Media Processing [Halfhill00,Khailany01]
- Computer Architecture [Rixner98]
- High Performance Graphics [Owens00, Purcell02, NVIDIA, ATI]

WireGL

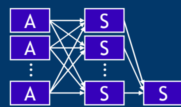
[Humphreys01]



WireGL Shortcomings

Sort-first

- Can be difficult to load-balance
- Screen-space parallelism limited
- Heavily dependent on spatial locality



Resource utilization

- Geometry must move over network every frame
- Server's graphics hardware remains underutilized

We need something more flexible

Chromium: General Approach

Replace system's OpenGL driver

- Industry standard API
- Support existing unmodified applications

Manipulate streams of API commands

- Alter/inject/discard commands and parameters
- Route commands over a network
- Render commands using graphics hardware

Graphics Stream Processing

Treat OpenGL calls as a stream of commands

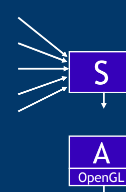
Form a DAG of stream transformation nodes

- Nodes are computers in a cluster
- Edges are OpenGL API communication

Each node has a *serialization* stage and a *transformation* stage

Stream Serialization

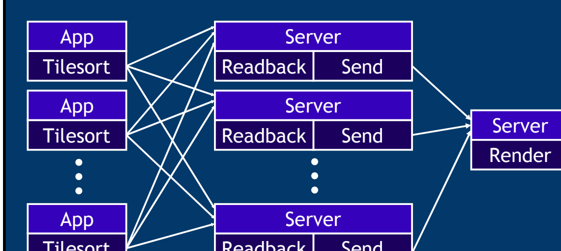
- Convert multiple streams into a single stream
- Context-switch between streams [Buck00]
- Constrain ordering using Parallel OpenGL extensions [Igehy98]
- Two kinds of serializers:
  - Network server:
  - Application:
    - Unmodified serial application
    - Custom parallel application



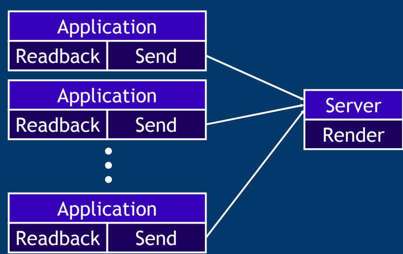
Stream Transformation

- Serialized stream is dispatched to "Stream Processing Units" (SPUs)
- Each SPU is a shared library
  - Exports the OpenGL interface
- Each node loads a *chain* of SPUs at run time
- Common usage: intercept a few OpenGL calls, pass all others to downstream SPU

Example: WireGL Reborn

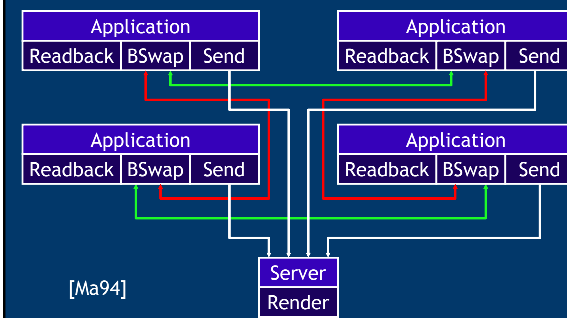


Example: Sort-Last

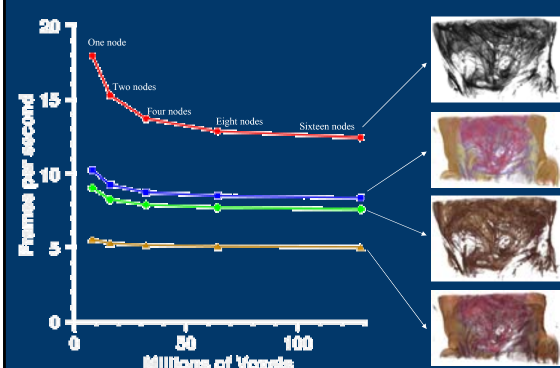


Application runs directly on graphics hardware
Same application can use sort-last or sort-first

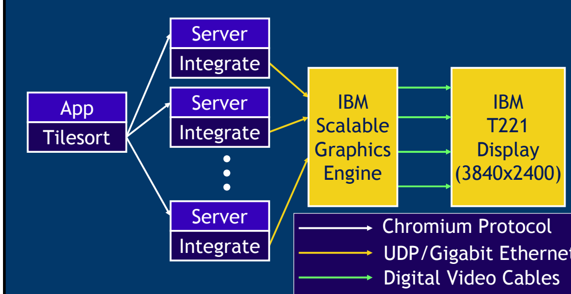
Sort-Last Binary Swap



Binary Swap Results



Example: UI Reintegration



CATIA Driving IBM's T221

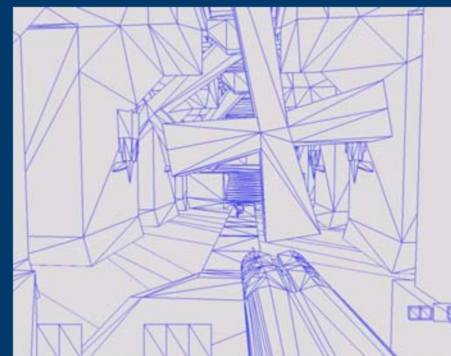
Jet engine nacelle model courtesy Goodrich Aerostructures
Chromium is the only practical way to drive the T221 with an existing application
Demonstrated at Supercomputing 2001

Example: Hidden-Line Drawing

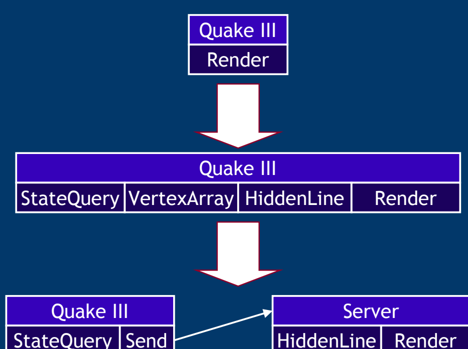
- Buffer a complete frame
- Play the frame back twice
- Wrinkles:
  - Vertex array pointers may not be valid at playback
  - State queries (e.g. `glGet`) must be handled immediately

A Hidden-line Style SPU

A Hidden-line Style SPU

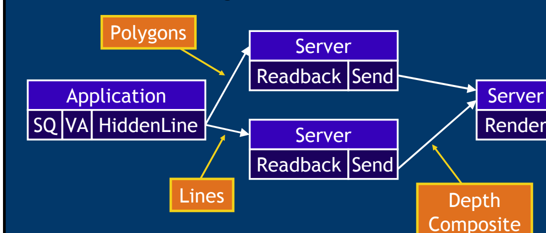


Demo



Is "HiddenLine" Really a SPU?

- Technically, no!
- Requires potentially unbounded resources
- Alternate design:



Future Directions

Taxonomy of non-invasive techniques
- Classify SPUs and algorithms
- Identify tradeoffs in design

End-to-end visualization system for 4D data
- Data management and load balancing
- Volume compression

Remote/Ubiquitous Visualization
- Scalable graphics as a shared resource
- Transparent remote interaction with (parallel) apps

Summary/Predictions

Manipulation of graphics streams is a powerful abstraction for cluster graphics
- Achieves both input and output scalability

Providing *mechanisms* instead of *algorithms* allows greater flexibility
- Data management algorithms can be built into a parallel application or embedded in a SPU

Flexible remote graphics will lead to a revolution in ubiquitous computing

Acknowledgements

- Pat Hanrahan
- Brian Paul and Alan Hourihane
- Ian Buck and Matthew Eldridge
- Chris Niederauer
- All the Chromium users
- DOE VIEWS grant #B504665

Try It Yourself

- Chromium is open-source
- Available from chromium.sourceforge.net
- Runs on:
  - Windows
  - Linux (tested on Intel and Playstation2)
  - IRIX
  - AIX
  - Solaris
  - HPUX
  - Tru64
  - Mac OS X (soon)

SPU Inheritance

The Readback and Render SPUs are related

- Readback renders everything except SwapBuffers

Readback *inherits* from the Render SPU

- Override parent's implementation of SwapBuffers
- All OpenGL calls considered "virtual"

Readback's SwapBuffers

```
void RB_SwapBuffers(void)
{
  self.ReadPixels( 0, 0, w, h, ... );
  child.Clear( GL_COLOR_BUFFER_BIT );
  child.BarrierExec( READBACK_BARRIER );
  child.RasterPos2i( tileX, tileY );
  child.DrawPixels( w, h, ... );
  child.BarrierExec( READBACK_BARRIER );
  child.SwapBuffers( );
}
```

Easily extended to include depth composite
All other functions inherited from Render SPU